IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Glen VAN DATTA et al.
Serial No.	:	10/700,798
Filed	:	November 3, 2003
For	:	PEER-TO-PEER RELAY NETWORK
Examiner	:	Ramy M. Osman
Art Unit	:	2157
Confirmation No.	:	6261

745 Fifth Avenue
New York, NY 10151

Mailing Label Number:
Date of Deposit:

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" Service under 37 CFR 1.10 on the date indicated above and is addressed to: **Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

(Typed or printed name of person mailing paper or fee)

(Signature of person mailing paper or fee)

DECLARATION UNDER 37 CFR 1.131

Mail Stop AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

I, Anthony Mai, hereby declare as follows:

1. I am a named inventor of the above-noted United States Patent Application 10/700,798, filed in the United States Patent and Trademark Office on November 3, 2003, with a

claim of priority under 35 U.S.C. 119(e) to Provisional Application 60/513,098, filed October 20, 2003.

2. I hereby declare I conceived and reduced to practice the invention defined by claim 24 ("the invention") of the above-noted application prior to April 9, 2002, the United States filing date of United States Patent 7,174,382 issued to Ramanathan et al. ("Ramanathan"), as demonstrated in the exhibits attached to this Declaration. My earlier conception and reduction to practice of my claimed invention is evidenced by the following statements:

3. Prior to April 9, 2002, I conceived of the invention of the present application as evidenced by Exhibit A, titled "Multi-Channel Multi-Party Audio Streaming Protocol" ("Protocol"), which was attached to an e-mail that I sent to G. Van Datta prior to April 9, 2002. Language in the e-mail portion of Exhibit A has been redacted to preserve attorney-client privileged information. Specific nomenclature in the Protocol has been redacted to preserve confidential information.

4. The Protocol discloses the elements recited in claim 24. In particular, the Protocol describes the method of joining (adding) a peer system to a peer-to-peer (P2P) system and a method of establishing a P2P network.

5. My invention was reduced to practice in a computer implementation as evidenced by the attached Exhibits B and C, which perform the functions recited by the elements recited in claim 24. These exhibits are source code that is proprietary to the assignee of the present invention; and such source code has been redacted to preserve the confidentiality of such source code.

6. Exhibit B is computer source code created prior to April 9, 2002. Exhibit B constructs and sends out communications packages, as well as receives and processes incoming

communication packages, pertaining to the forming and maintenance of the relay grid. Exhibit B describes the data packages that the relay grid tries to relay. Portions of Exhibit B have been redacted to preserve confidential information.

7. Exhibit C is computer source code created prior to April 9, 2002. Exhibit C manages the features in Exhibit B as well as manages the high level application requests. Exhibit C generates and processes the message packages that are used to implement the invention. Exhibit C also accepts incoming and outgoing audio data streams and processes the data streams in proper data packages Exhibit C utilizes and manages Exhibit B to allow each client to interact with each other using pre-defined message packages in order to connect to each other and form the relay grid described in the invention. Function calls from Exhibit C are reproduced in Exhibits C1-C8 and are explained in more detail herein as necessary. Portions of Exhibits C and C1-C8 have been redacted to preserve confidential information.

8. The function call on page 10 of Exhibit C and reproduced as Exhibit C1 causes the code to start the process to construct a relay grid, which implements the element "adding a peer system to a peer-to-peer relay network," recited in claim 24.

9. The function call on page 11 of Exhibit C and reproduced as Exhibit C2 causes the code to process any incoming network package and decide further processing depending on the package, which implements "opening a connection between a server and a joining peer system," recited in claim 24.

10. The function call on page 24 of Exhibit C and reproduced as Exhibit C3; the function call on page 25 of Exhibit C and reproduced as Exhibit C4; the function call on page 26 of Exhibit C and reproduced as Exhibit C5; and the function call on page 27 of Exhibit C and reproduced as Exhibit C6; cause the code to allow top application layer code to obtain

information about existing channels (relay grid) and clients who have joined in each channel, which implements "providing grid information to said joining peer system indicating one or more established peer-to-peer relay networks," recited in claim 24.

11. The function call on page 22 of Exhibit C and reproduced as Exhibit C7 causes the code to cause the local client to join a relay grid, which implements "receiving a grid selection from said joining peer system indicating a selected peer-to-peer relay network, wherein said selected peer-to-peer relay network has one or more member peer systems," recited in claim 24.

12. The function call on page 27 of Exhibit C and reproduced as Exhibit C8 causes the code to provide bookkeeping of the network address of individual member peer systems to the underlying implementation of the peer relay system, which implements "providing network addresses of each of said one or more member peer systems to said joining peer system," recited in claim 24.

13. The function call on page 22 of Exhibit C and reproduced as Exhibit C7 causes the code to enable a local client to join a relay grid, which implements "receiving a connection update from said joining peer system indicating to which member peer systems said joining peer system is connected," recited in claim 24.

14. The function call on page 22 of Exhibit C and reproduced as Exhibit C7 causes the code to start a sequence of actions and message exchanges, which implements "wherein each member peer system is connected to a number of other member peer systems that is less than or equal to a connection limit and each member peer system stores a set of one or more relay rules for relaying data to the other member peer systems connected to that member peer system," recited in claim 24.

15. As evidenced by attached Exhibits A through C, every element of my claimed invention was reduced to practice prior to April 9, 2002.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

[illegible] Sep. 19, 2008

Signature of Declarant
Anthony Mai

Date

Print or Typed of Declarant

EXHIBIT A

From: [redacted]
Sent: [redacted]
To: [redacted]
Subject: [redacted]

Attachments: [redacted]

----- [redacted] -----

[redacted]

To

[redacted]

cc

[redacted]

Subject

[redacted]

[redacted]

Anthony Mai
Sony Computer Entertainment America
http://www.scea.com
----- [redacted] -----

Anthony Mai/SDPD/SCEA

[redacted]

To
Glen Van
Datta/SDPD/SCEA

cc

Subject
The document

Glen:

Here is the doc file attackment.

Anthony Mai
Sony Computer Entertainment America
http://www.scea.com (See attached file: [redacted].doc)

Multi-Channel Multi-Party Audio Streaming Protocol

Introduction

Audio streaming in the online game scenery is different from conventional VoIP application in a number of ways. First, conventional VoIP system has only one data source. It may have one data target, like in the case of internet telephone, or it may have one server and multiple data targets, like in the case of internet radio or other broadcast steaming.

In the online game scenery, there could be multiple data sources (each player in a game may speak), and there could also be multiple data targets (each player in a game may also listen.) And there may not even be a central server to receive and re-distribute all the audio data.

Due to network bandwidth limitation, a multi-channel Multi-party audio streaming protocol must be designed to allow multiple players to talk with each other over the network.

Assumption of the protocol:

The following assumptions will be made:

1. There will always be one [redacted] players in a game. [redacted]

2. [redacted] maintains a list of all the available audio channels, or audio room [redacted] authorization of each player in each room to speak. Each individual player [redacted] keeps a copy of the audio room list.

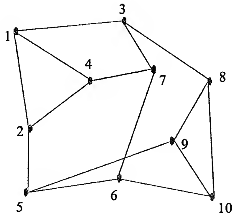
3. The network grid. Each [redacted] will be connected to no more than [redacted] other [redacted] directly. Any data initially received by a [redacted] from one of the [redacted] it connects to will be forwarded to the two other [redacted]. Repeated data is not forwarded. By this mechanism, data from any one [redacted] can eventually spread to all [redacted] in the grid.

The network grid

The grid limits each [redacted] network bandwidth requirement (since it only needs to communicate with [redacted] other [redacted]) while allowing data from any single [redacted] to quickly spread to every other [redacted] in the grid, using [redacted] sockets.

There will be a network grid for the channel 0, or room #0. All [redacted] are connected to this grid. This will be used for none-audio control messages.

There will be [redacted] for each audio room. Each [redacted] can optionally join a specific audio room. But each [redacted] can join no more than one audio room at a time.



Each [redacted] is connected to 3 other [redacted], forming a grid

Within each audio room grid, [redacted] Any [redacted] wishing to speak should wait until the speaker has finished or paused. If multiple players try to speak, [redacted]

Establishment of the audio room network grid

Any [redacted] can create an audio room and this event is broadcast to every one in the game, through the channel 0 grid. The [redacted] that created the audio room becomes the first node of the network grid. It will notice that it has three arms of connection, none occupied yet.

Any [redacted] wishing to join an audio room broadcast a message through the channel 0 grid. Every [redacted] within the grid then sends the new [redacted] a welcome message, specifying whether they have a free arm of connection available or not.

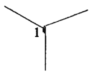
The new [redacted] first takes offers of connection from existing [redacted] who has a free arm of connection. And then request connection with the very first [redacted] who greeted it [redacted] Upon such request, the existing [redacted] [redacted] one of its connections and connects with the new [redacted].

Any [redacted] with a [redacted] connection will then [redacted] the availability of connection on channel 0. The [redacted] who also have free connection arm will respond to [redacted] establishes a connection.

Establishment of the channel 0 grid

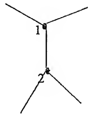
Channel 0 grid is the network grid that every [redacted] So it is important that when each [redacted] joins the game, [redacted]

1. First [redacted]
   When the first [redacted] creates a new game session, there are no other [redacted]. So [redacted] three grid connections of the first [redacted] is available.



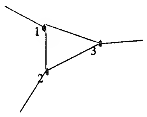
First [redacted] in a game

Second [redacted]



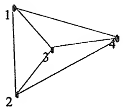
Second [redacted] joins the game. The game server notifies first [redacted] about the second [redacted]. The first [redacted] sends a welcome message [redacted] and they connect.

Third [redacted] joins



Third [redacted] joins the game. The game server notifies first 2 [redacted] about the third [redacted]. The first 2 [redacted] send a welcome message [redacted] and they connect.

Fourth [redacted] joins



Fourth [redacted] joins the game. [redacted]

The new [redacted] joining protocol:

1. The game server sends the new [redacted] info to all existing [redacted] within a game.
2. Each existing [redacted] sends a welcome UDP message to the new [redacted], indicating whether it has any spare connection arm available.
3. The new [redacted] respond [redacted] the [redacted] directly.
4. The [redacted] that receives the connection request message [redacted] a connection accepted message back. And the connection is established.
5. If the new [redacted] still has 2 or more connection arms unused, it further sends a connection request message to the first existing [redacted] who sent a welcome message with no available connection arm.
6. Upon connection requests from the new [redacted], an existing [redacted] would [redacted] break one of the connection arms, and then sends a connection accepted message back to the new [redacted].

Maintaining the grid

[redacted] could drop out of the network grid [redacted] So it is important that the grid can [redacted] connection.

1. Each pair of connected [redacted] to each other periodically.
   - [redacted] indicates that a connection is still alive.
2. If [redacted] A is informed of disconnection [redacted], or [redacted] not been received [redacted] the [redacted] marks that connection arm as free, and send out a connection arm available message, on channel 0.
3. When [redacted] B receives a connection arm available message, and it has a free connection arm, it responds by sending a connection request message.
4. When [redacted] A receives a connection request message, [redacted] connection accepted [redacted] And the two is connected.

Dropping out of the grid

If a [redacted] intend to quit the game, it should post a message to all [redacted] it connects to indicating that it is quitting.

When [redacted] time interval from a connected [redacted], [redacted] the connection no longer exist, and [redacted] through the existing arms of connection that it has a free arm of connection available. Any other [redacted] who also has a free arm of connection [redacted] the two can connect.

Transferring data within the grid

The network grid is established to allow all [redacted] within the grid to share information with minimal network load for each individual [redacted].

A [redacted] so when a [redacted] receives the package, it knows whether it has already received the package or not. If the [redacted] hasn't [redacted] package to the other two [redacted] it connects to, except for the one that it just received the package from. If a [redacted] [redacted] preventing the package from further circulation within the grid.

[redacted] no repeated packages received, and chances of lost package will be extremely low, since each [redacted] is [redacted] and the packaged could be forwarded to the [redacted] [redacted] This way, the network [redacted]

Transferring audio data within the grid

Each package of audio data [redacted]

When each [redacted] within the [redacted]

Each [redacted] decide whether it can speak or not according to the following rules:

1. If no [redacted] package was received within certain time period (like 1-2 seconds), it can start to speak.
2. If one or [redacted] that means the previous speaker [redacted] so the [redacted] can start to speak.
3. If after a [redacted] starts to speak, it receives a [redacted] from a different [redacted], a collision has occurred. Under such circumstance the [redacted] should immediately stop speaking, and wait for the next opportunity to speak.
4. When the [redacted] can start [redacted] there [redacted] that the human player can start to speak. The player may or may not speak. If the player speaks, it will be [redacted] and the [redacted] will begin [redacted] to the network grid.
5. When a [redacted] speaks, [redacted] and a silence package will be send out, allowing other [redacted] to speak, while the local [redacted] will wait for next opportunity to speak.
6. If a [redacted] speaks for [redacted], it will be forcefully stopped, allowing other [redacted] an opportunity to start speaking.

EXHIBIT B

```
/******************************************************************************
**
*
* [redacted]
*
*******************************************************************************
*/

/******************************************************************************
**
*
*  [redacted]
*
*  Description:       The implementation of the network grid class.
*
*
*  [redacted]       Anthony Mai (am) anthony_mai@playstation.sony.com
*
*  [redacted]
*
*  [redacted]
*
*******************************************************************************
*/

#if [redacted]IN32)
#inc[redacted]b.h>
#inc[redacted]y.h>
#end[redacted]32)

#in[redacted]g.h>
#in[redacted]o.h"
#in[redacted]l.h"
#in[redacted]d.h"
#in[redacted]o.h"
#in[redacted]e.h"
#in[redacted]r.h"
#in[redacted]f.h"


#if[redacted]32)
sta[redacted]d();
#e[redacted]f  .

#de[redacted]nds
#de[redacted]nds
#de[redacted] of
s[redacted]h.
```

```
#if[redacted]2)
sta[redacted]d()
{
        sta[redacted]89;

        las[redacted]011;

        ret[redacted]ff);
}
#e[redacted]f

/*******************************************************************************
**      .
* Function:         [redacted]
*
* Description:    Invalidate a specific connection by setting the status flag
*                to STATUS_INVALID and nollify the IP and Port.
*
* Returns:       [redacted]
*******************************************************************************
*/
vo[redacted]te()
{
     sta[redacted]LID;
     IP [redacted]= 0;
     Por[redacted]= 0;
     las[redacted]= 0;
     nPi[redacted]= 0;
}


/*******************************************************************************
**
* Function:         [redacted]
*
* Description:      [redacted] class constructor
*
* Returns:       [redacted]
*******************************************************************************
*/
CG[redacted]id
(
     CA[redacted]ent,
     RT[redacted]nel
) :
     m_p[redacted]nt),
     m_C[redacted]el),
     m_p[redacted]L),
     m_n[redacted]L),
     m_n[redacted](0),
     m_L[redacted]e(0),
     m_b[redacted](0),
     m_C[redacted](0),
     m_C[redacted](0),
     m_L[redacted](0)
{
     RT[redacted]s();
```

```
    m_N[redacted]00;
    m_L[redacted]ime;
    mem[redacted]on));
    for[redacted]++)
    {
        m_C[redacted]ime;
    }
    mem[redacted]ng));
}

/***************************************************************************
**
* Function:        [redacted]
*
* Description:     [redacted] class destructor
*
* Returns:    [redacted]
***************************************************************************
*/
CG[redacted]id()
{
}

/***************************************************************************
**
* Function:        [redacted]
*
* Description: [redacted]
*
*
* Returns:    [redacted]
***************************************************************************
*/
RT[redacted]ce()
{
    ret[redacted]++;
}

/***************************************************************************
**
* Function:        [redacted]e
*
* Description:     Send a package to all [redacted] this grid connects to, [redacted]
*                  [redacted]
*
* Returns:     [redacted]
***************************************************************************
*/
AUD[redacted]ge
(
    RT_[redacted]IP,
    RT_[redacted]rt,
    AUD[redacted]dr,
        con[redacted]er,
        RT_[redacted]ze
)
{
        RT_[redacted]       i;
        RT_[redacted]ze;
        AUD[redacted]OR;
        RT_[redacted]ce;
        RT_[redacted]ZE];

        // Fi[redacted]ge.
        mem[redacted]dr));
        nSi[redacted]dr);
        swi[redacted]pe)
        {
        c[redacted]M:
        c[redacted]M:
        c[redacted]0:
            // We[redacted]ges.
            nAu[redacted]();
            mem[redacted]e));
            nS[redacted]e);
            br[redacted];
        de[redacted]:
            b[redacted]k;
        }

        mem[redacted]ize);
        nSi[redacted]ze;

        for[redacted]++)
        {
            if ([redacted] &&
                ((m_[redacted]) ||
                (m_C[redacted]t)) )
            {
                if [redacted]dTo(
                    tm[redacted]f,
                    nS[redacted],
                    m_C[redacted]IP,
                    m_C[redacted]ort
                    ) )
                {
                    ret[redacted]ROR;
                }
                e[redacted]
                {
                    re[redacted]IC;
                    b[redacted]k;
                }
            }
        }

        re[redacted]et;
}
```

```
/*****************************************************************************
**
* Function:          [redacted]
*
* Description:    Set the standard package header for usage.
*
* Returns:  [redacted]
*****************************************************************************
*/
voi[redacted]er
(
    AUD[redacted]r,
    RT_[redacted]e,
    RT_[redacted]n
)
{
    // We [redacted] bit.
    pH[redacted]TOR);
    pH[redacted]Len;
    pH[redacted]();
    pH[redacted]nel;
    pH[redacted]lIP;
    pH[redacted]rt;
}


/*****************************************************************************
**
* Function:          [redacted]
*
* Description:    The update function. When called, [redacted]controled processings
*                are done at appropriate times.
*
* Returns:       [redacted]e.
*****************************************************************************
*/
AUD[redacted]e()
{
    RT_[redacted] i;
    RT_[redacted]Ms();

    if [redacted]ime) >
JO[redacted]T))
    {
        for[redacted]i++)
        {
            if [redacted]LID)
            {
                b[redacted]k;
            }
        }
        if (i [redacted] [redacted]ONS)
        {
            AUD[redacted]Hdr;
            CLI[redacted]st;

            con[redacted]el;
        con[redacted]lIP;
        con[redacted]ort;

        m_C[redacted]NG;
        m_C[redacted]tIP;
        m_C[redacted]ort;
        m_C[redacted];
        m_C[redacted] 0;

        Set[redacted]r(
            &se[redacted]dr,
            MSG[redacted]CT,
            siz[redacted]est)
            );

        // Do [redacted]ow?
        //se[redacted]nel;

        m_p[redacted]ge(
            m_L[redacted]IP,
            m_L[redacted]ort,
            &se[redacted]dr,
            &co[redacted]st,
            siz[redacted]st)
            );
    }
    m_L[redacted]e = 0;
    mem[redacted]ng));
}

for[redacted]i++)
{
    if ([redacted]ED)
    {
        if ([redacted]me))
            < PI[redacted]AL )
        {
            // We [redacted] OK.
        }
        el[redacted] < 3)
        {
            // Let[redacted]live
            AUD[redacted]dr;

            m_C[redacted]ime;

            Set[redacted] 0);
            m_p[redacted]ge(
                m_C[redacted].IP,
                m_C[redacted]rt,
                &se[redacted]Hdr,
                N[redacted],
                [redacted]
                );
        }
        e[redacted]
        {
```

```
                    // The[redacted]the
g[redacted]d.
                    AUD[redacted]dr;
                    CLI[redacted]it;

                    Set[redacted]IT,
si[redacted]it));
                    // We[redacted]ere.
                    sen[redacted]OR;
                    sen                                            IP;
                    sen                                                      ort;

                    cli[redacted]it);
                    cli                                            IP;
                    cli                                                      rt;

                    Sen[redacted]ge(
                        cli[redacted]IP,
                        cli[redacted]rt,
                        &se[redacted]dr,
                        &cl[redacted]it,
                        siz[redacted]it)
                        );

                    if (m_[redacted] 0)
                    {
                        Re[redacted]r(
                            cli[redacted]IP,
                            cli[redacted]ort
                            );
                    }

                    m_p[redacted]r(
                        cli[redacted]IP,
                        cli[redacted]ort
                        );
                }
            }
        }

    ret[redacted]OR;
}


/*****************************************************************************
**
* Function:          [redacted]
*
* Description:       Process the audio data package. The data [redacted]
*                   [redacted]fetched by the application.
*
* Returns:           [redacted]
*****************************************************************************
*/
AUD[redacted]ack
(
    AUD[redacted]Hdr,
        RT_[redacted]mIP,
        RT_[redacted]rt,
        con[redacted]fer,
        RT_[redacted]ze
)
{
        RT_[redacted]   i;
        RT_[redacted]nce;
        RT_[redacted]eMs();

        if [redacted]SM)
        { // We[redacted] GSM
            ret[redacted]RIC;
        }

        // Fir[redacted]age
        for [redacted]i++)
        {
            if (m[redacted]TED)
            {
                if [redacted] &&
                    (m_C[redacted]ort) )
                {
                    // The [redacted] is
[redacted]e.
                    m_C[redacted]ime;
                    m_C[redacted]= 0;
                }
                els[redacted]IP) &&
                    (m_[redacted]rt) )
                {
                    // We [redacted]tor
                }
                e[redacted]e
                {
                    m_p[redacted]To(
                        pB[redacted]r,
                        cb[redacted]e,
                        m_C[redacted].IP,
                        m_C[redacted]rt
                        );
                }
            }
        }

       pBu[redacted]Hdr);
       cbS[redacted]Hdr);

       mem[redacted]ce));
       pBu[redacted]nce);
       cbS[redacted]nce);

       m_p[redacted]ce(
           pHd[redacted]IP,
           pHd[redacted]rt,
           nAu[redacted]ce,
           pBu[redacted]r,
        cb[redacted]e
        );

    m_La[redacted]me;
    if [redacted] 0)
    {
        m_C[redacted]rIP;
        m_C[redacted]ort;
    }
    e[redacted]
    {
        m_C[redacted]= 0;
        m_C[redacted]t = 0;
    }

    ret[redacted]OR;
}


/*****************************************************************************
**
* Function:           [redacted]k
*
* Description:        Process [redacted] data control packages. And respond accordingly.
*
* Returns:          [redacted]e.
*****************************************************************************
*/
AUD[redacted]ack
(
    AUD[redacted]dr,
    RT_[redacted]IP,
    RT_[redacted]rt,
    con[redacted]fer,
    RT_[redacted]ize
)
{
    int[redacted]i;
    AU[redacted]dr;
    AU[redacted]ROR;
    RT_[redacted]Ms();
    RT_[redacted]lse;
    un[redacted]n {
        con[redacted]p;
        con[redacted]Join;
        con[redacted]uest;
        con[redacted]nel;
        con[redacted]ing;
        con[redacted]ept;
        con[redacted]ect;
        con[redacted]ect;
        con[redacted]ect;
        con[redacted]Info;
        con[redacted]nfo;
        con[redacted]nel;
        con[redacted]nel;
        con[redacted]uit;
```

```
};

p =[redacted]HDR));
// Fir[redacted] do,
// for[redacted]ngs.

swi[redacted]ype)
{
cas[redacted]NE:
cas[redacted]NG:
cas[redacted]NG:
    bF[redacted]se;
    [redacted]k;
cas[redacted]AT:
cas[redacted]IN:
    bF[redacted]rue;
    br[redacted];
cas[redacted]ECT:
    bF[redacted]ue;
    br[redacted];
cas[redacted]NG:
    bF[redacted]se;
    [redacted]k;
cas[redacted]NEL:
    bF[redacted]rue;
    br[redacted];
c[redacted]O:
    [redacted]e;
    [redacted];
cas[redacted]NEL:
cas[redacted]NEL:
cas[redacted]IT:
    bF[redacted]rue;
de[redacted]t:
    br[redacted];
}

for [redacted] i++)
{
    if [redacted]TED)
    {
        if [redacted] &&
            (m_C[redacted]rt) )
        {
            // The[redacted]y is
a[redacted]e.
            m_C[redacted]ime;
            m_C[redacted]= 0;
        }
        els[redacted]) &&
            (m_C[redacted]ort) )
        {
            // We d[redacted]tor
        }
        els[redacted]sg)
        {
            m_p[redacted]dTo(
                    pB[redacted]r,
                    cb[redacted],
                    m_C[redacted]IP,
                    m_C[redacted]rt
                    );
            }
        }
    }

    // Now[redacted]sage.
    swi[redacted]pe)
    {
    cas[redacted]NE:
        br[redacted]

    cas[redacted]NG:
        {
            Set[redacted], 0);

            m_p[redacted]ge(
                fr[redacted]P,
                fr[redacted]rt,
                &s[redacted]dr,
                NU[redacted],
                0
                );
        }
        br[redacted];

    cas[redacted]G:
        br[redacted];

    cas[redacted]AT:
        b[redacted]k;

    cas[redacted]IN:
        {
            CPl[redacted]er;

            pNe[redacted]er(
                pHd[redacted]IP,
                pHd[redacted]rt,
                m_C[redacted]l
                );
            Add[redacted]r);
            ret[redacted]g(
                p[redacted]d[redacted]IP,
                pHd[redacted]ort,
                t[redacted]
                );
        }
        b[redacted]k;

    cas[redacted]CT:
        {
            ret[redacted]ng(
                pHd[redacted]IP,
                pHd[redacted]rt,
                fa[redacted]
                );
        }
        b[redacted]k;

    cas[redacted]NG:
        {
            CPl[redacted]er;
            CLI[redacted]st;

            pPl[redacted]er(
                pHd[redacted]IP,
                pHd[redacted]rt,
                pGr[redacted]el
                );

            if (m[redacted] 0)
            {
                Add[redacted]r);

                if [redacted] 0)
                {
                    CGr[redacted]el);
                    if [redacted]LL)
                    {
                        REQ[redacted]fo;

                        // Add[redacted]nel
                        pGr[redacted]el);

                        // We [redacted]nel
                        req[redacted]el;

                        Set[redacted]r(
                            &s[redacted]dr,
                            MSG[redacted]FO,
                            siz[redacted]fo)
                            );

                        m_p[redacted]ge(
                            pHd[redacted]IP,
                            pHd[redacted]rt,
                            &se[redacted]dr,
                            &re[redacted]fo,
                            siz[redacted]fo)
                            );
                    }
                    pGr[redacted]er);
                }
            }

            for [redacted]++)
            {
                if ((m_C[redacted]ED) &&
                    (m_C[redacted]P) &&
                    (m_C[redacted]t) )
                {
                    b[redacted]k;
                }
            }
            if ([redacted]NS)
            {
                // We [redacted]nt
                br[redacted];
            }
            for [redacted]i++)
            {
                if (m_[redacted]ID)
                {
                    br[redacted];
                }
            }
            if ([redacted]NS)
            { // We [redacted]dy
                br[redacted];
            }
            if ([redacted]s > 0)
            {
                con[redacted]nel;
                con[redacted]lIP;
                con[redacted]ort;

                m_C[redacted]NG;
                m_C[redacted]rIP;
                m_C[redacted]ort;
                m_C[redacted]me;
                m_C[redacted]= 0;

                Set[redacted]r(
                    &se[redacted]dr,
                    MSG[redacted]CT,
                    siz[redacted]st)
                    );

                m_p[redacted]ge(
                    pHd[redacted]IP,
                    pHd[redacted]rt,
                    &se[redacted]dr,
                    &co[redacted]st,
                    siz[redacted]st)
                    );
            }
            e[redacted]
            {
                m_L[redacted]me;
                m_L[redacted]g;
            }
        }
        br[redacted];

    cas[redacted]CT:
        {
            CLI[redacted]pt;
        for [redacted]++)
        {
            if [redacted]ED)
            {
                // Fou[redacted]nch.
                br[redacted];
            }
        }

        if [redacted]NS)
        {
            // If w[redacted]ted
            CLI[redacted]ct;

            i =[redacted]NS;
            cli[redacted]el;
            cli[redacted]IP;
            cli[redacted]nt-
>m_[redacted]t;

            Set[redacted]r(
                &se[redacted]dr,
                MSG[redacted]CT,
                siz[redacted]ct)
                );

            m_[redacted]e(
                m_[redacted]P,
                m_[redacted]t,
                &s[redacted]r,
                &c[redacted]t,
                si[redacted]t)
                );

            m_C[redacted]D;
            m_C[redacted] 0;
            m_C[redacted] 0;
            m_C[redacted]e = 0;
        }

        m_C[redacted]IP;
        m_C[redacted]rt;
        m_C[redacted]ED;

        con[redacted]el;
        con[redacted]IP;
        con[redacted]rt;

        Se[redacted]Hdr,
            MS[redacted]CT,
            siz[redacted]pt)
            );

        m_p[redacted]e(
            pHd[redacted]IP,
            pHd[redacted]rt,
                  &se[redacted]dr,
                  &co[redacted]pt,
                  s[redacted]z[redacted]pt)
                  );
        }
        br[redacted];

    cas[redacted]CT:
        {
            for [redacted]++)
            {
                if ([redacted] &&
                    (m_C[redacted] &&
                    (m_C[redacted]rt) )
                {
                    // Foun[redacted]nch.
                    br[redacted];
                }
            }

            if [redacted]ONS)
            {
                m_C[redacted]ED;
                m_C[redacted]ime;
                m_C[redacted]= 0;
            }
            e[redacted]
            {
                // Tell[redacted]ion.
                CLI[redacted]ect;

                cli[redacted]nel;
                cli[redacted]lIP;
                cli[redacted]nt-
>m_L[redacted]t;

                Set[redacted]r(
                    &se[redacted]dr,
                    MSG[redacted]CT,
                    siz[redacted]ct)
                    );

                m_p[redacted]ge(
                    pHd[redacted]P,
                    pHd[redacted]t,
                    &se[redacted]dr,
                    &cl[redacted]ct,
                    siz[redacted]ect)
                    );
            }
        }
        br[redacted];

    cas[redacted]CT:
        // To [redacted]e.
        br[redacted];
        cas[redacted]CT:
          {
              for [redacted]i++)
              {
                  if ([redacted]) &&
                      (m_C[redacted]ort) )
                  {
                      CLI[redacted]t;

                      m_C[redacted]D;
                      m_C[redacted] = 0;
                      m_C          = 0;
                      m_C          = 0;

                      may[redacted]el;
                      may[redacted]IP;
                      may          nt-
>m_[redacted]t;

                      Set[redacted]Hdr,
                          MSG[redacted]CT,
                          siz[redacted]ct)
                          );

                      Sen[redacted]e(
                          pHd[redacted]P,
                          pHd         t,
                          &se         r,
                          &ma         ct,
                          siz         t)
                          );

                      b[redacted]k;
                  }
              }
          }
          b[redacted]k;

        cas[redacted]EL:
          {
              CGr[redacted]nel);
              if [redacted]LL)
              {
                  // Chann[redacted]it
              }
              e[redacted]
              {
                  CPl[redacted]r;

                  pPl[redacted]r(
                      pHd[redacted]IP,
                      pHd         rt,
                      pCr         l
                      );

                  pGr[redacted]nel);
```

```
                    str[redacted]e,
siz[redacted]me));
                    pGr[redacted]x00;

                    pGr[redacted]er);
                }
            }
            b[redacted]k;

        cas[redacted]NFO:
            {
                CGr[redacted]nfo-
>nC[redacted]1);

                if [redacted]d)
                {
                    CHA[redacted]o;

                    cha[redacted]fo);
                    cha                el;
                    str       (
                        ch[redacted]me,
                        pGr[redacted]e,
                        siz[redacted]me)
                        );
                    cha[redacted]me) -
1][redacted]0;

                    Set[redacted]FO,
siz[redacted]o));
                    m_p[redacted]e(
                        pHd[redacted]IP,
                        pHd[redacted]rt,
                        &se[redacted]dr,
                        &ch[redacted]fo,
                        siz[redacted]fo)
                        );
                }
            };
            br[redacted];

        cas[redacted]FO:
            {
                CG[redacted]el);

                if [redacted]d)
                {
                    st[redacted](
                        pGr[redacted]me,
                        pCh[redacted]me,
                        siz[redacted]me)
                        );
                }
            }
            br[redacted];

        cas[redacted]EL:
```

```
        if [redacted]l > 0)
        {
            CGr[redacted]d;
            CPl[redacted]er(
                pHd[redacted]IP,
                pHd[redacted]rt
                );

            if [redacted]L)
            {
                pPl[redacted]r(
                    pHd[redacted]IP,
                    pHd[redacted]rt,
                    pJo[redacted]el
                    );
            }
            els[redacted]l > 0)
            {
                if [redacted]l)) !=
[redacted]L)
                {
                    pGr[redacted]rt);
                }
            }

            pGr[redacted]el);

            if [redacted]LL)
            {
                br[redacted];
            }
            pGr[redacted]r);
            pPl[redacted]el);

            if [redacted]el)
            {
                ret[redacted]rt,
[redacted]e);
            }
        }
        br[redacted];

    cas[redacted]EL:
        if [redacted]l > 0)
        {
            CPl[redacted]er(
                pHd[redacted]IP,
                pHd[redacted]rt
                );
            CGr[redacted]el);

            if ([redacted]LL))
            {
                br[redacted];
            }

            pGr[redacted]rt);
            pPl[redacted]l(0);

            if [redacted]el)
            {
                AUD[redacted]on(
                    pHd[redacted]P,
                    pHd[redacted]rt
                    );

                if [redacted]L)
                {
                    pCo[redacted]();
                    Inv[redacted]();
                }
            }
        }
        br[redacted];

    cas[redacted]T:
        {
            m_p[redacted]rt);
        }
        br[redacted]k;

    de[redacted]t:
        b[redacted]k;
    }

    re[redacted]t;
}
// END [redacted]ck


/*****************************************************************************
**
* Function:         [redacted]
*
* Description:      Check to see if we can speak in this grid.
*
* Returns:          [redacted]
*
*****************************************************************************
*/
AUD[redacted] [redacted]wed()
{
    Upd[redacted]s();
    if [redacted]d)
    {
        ret[redacted]OR;
    }
    e[redacted]
    {
        ret[redacted]IC;
    }
}
```

```
/***************************************************************************
**
* Function:        [redacted]
*
* Description:    Update [redacted] status flag. [redacted]
*                 [redacted]
*
* Returns:      [redacted]
***************************************************************************
*/
void [redacted]()
{
    RT[redacted]i;

    // Do [redacted]ner?
    for ([redacted]++)
    {
        if ([redacted]ED)
        {
            br[redacted];
        }
    }
    if ([redacted]NS)
    {
        // No [redacted]us.
        m_b[redacted] 0;
    }
    els[redacted] &&
        ((m_C[redacted]P) ||
        (m_C[redacted]) )
    {
        // The[redacted]ing
        if ([redacted])) >
SPE[redacted]ME)
        {
            // Whoe[redacted]pped.
            m_C[redacted] 0;
            m_C[redacted]= 0;
            m_b[redacted] 1;
        }
        e[redacted]
        {
            m_b[redacted] = 0;
        }
    }
    e[redacted]
    {
        // No o[redacted]ing.
        m_b[redacted]d = 1;
    }

    if (!m[redacted]d)
    {
        // Mak[redacted]ce 0.
        m_p[redacted]e();
    }
}
```

```
/*************************************************************************
**
* Function:        [redacted]
*
* Description:     Send out a greeting message to a specific [redacted]
*
* Returns:        [redacted]
*************************************************************************
*/
AUD[redacted]ing
(
    RT_[redacted]IP
    RT_[redacted]ort
    RT_[redacted]le
)
{
    RT_[redacted]        i;
    AUD[redacted]dr;
    CLI[redacted]ng;

    gre[redacted]IP;
    gre[redacted]rt;
    gre[redacted]el;
    gre[redacted]= 0;
    gre[redacted]ng);

    for [redacted]i++)
    {
        if ([redacted]ID))
        {
            gre[redacted]++;
        }
    }

    if (b[redacted] 0))
    {
        Set[redacted]ing));

        ret[redacted]age(
            nD[redacted]P,
            nD[redacted]t,
            &s[redacted]r,
            &g[redacted]g,
            si[redacted]g)
            );
    }
    e[redacted]
    {
        ret[redacted]OR;
    }
}


/*************************************************************************
**
```

```
* Function:          [redacted]
*
* Description:    Return pointer to an [redacted] that matches the IP & Port.
*
* Returns:       [redacted]
*****************************************************************************
*/
AUD[redacted]ion
(
    RT_[redacted]P,
    RT_[redacted]t
)
{
    RT_[redacted]i;

    for ([redacted]++)
    {
        if ([redacted]&&
            (m_C[redacted]t) )
        {
            ret[redacted]i]);
        }
    }

    ret[redacted]L;
}


/*****************************************************************************
**
* Function:          [redacted]
*
* Description:    Sends a message that [redacted] connection on ourself.
*
* Returns:       [redacted]
*****************************************************************************
*/
AUD[redacted]t()
{
    AUD[redacted]dr;
    CLI[redacted]ct;

    may[redacted]el;
    may[redacted]IP;
    may[redacted]rt;

    Set[redacted]dr,
        MSG[redacted]CT,
        siz[redacted]t)
        );

    ret[redacted]e(
        m_p[redacted]P,
        m_p[redacted]t,
        &se[redacted]r,
        &ma[redacted]t,
        siz[redacted]t)
```

```
            );
}

voi[redacted]t()
{
    RT_[redacted]i;
    AUD[redacted]dr;
    CLI[redacted]it;

    Set[redacted]it));

    cli[redacted]it);
    cli[redacted]lIP;
    cli[redacted]ort;

    Sen[redacted]e(
        cli[redacted]IP,
        cli[redacted]rt,
        &se[redacted]r,
        &cl[redacted]t,
        siz[redacted]it)
        );

    m_P[redacted]();
    for [redacted]i++)
    {
        m_C[redacted]e();
    }
}

AUD[redacted]yer
(
    ENU[redacted]nc,
    uns[redacted]a
)
{
    RT_[redacted];

    for [redacted]i++)
    {
        if [redacted]ED)
        {
            CPl[redacted]er(
                m_C[redacted]IP,
                m_C[redacted]rt
                );

            if (p[redacted]L)
            {
                CLI[redacted]o;

                cli[redacted].IP;
                cli[redacted]rt;
                cli[redacted]el;
                if [redacted]fo))
                {
                    b[redacted];
                }
            }
        }
    }

    ret[redacted]OR;
}

voi[redacted]on()
{
    RT_[redacted]i;

    for ([redacted]i++)
    {
        if [redacted]ED)
        {
            m_C[redacted]te();
        }
    }
}

voi[redacted]er
(
    CPl[redacted]er
)
{
    m_P[redacted]er);
}

voi[redacted]er
(
    RT_[redacted]IP,
    RT_[redacted]rt
)
{
    AUD[redacted]rt);

    if [redacted]LL)
    {
        pCo[redacted]e();
        Inv[redacted]t();
    }

    m_P[redacted]rt);
}

voi[redacted]er
(
    CLI[redacted]er
)
```

```
{
    pSp[redacted]el;
    pSp[redacted]00;

    if [redacted]rt)
    {
        if ([redacted]e)) >
SP[redacted]ME)
        {
            // Whoe[redacted]ped.
            m_C[redacted]= 0;
            m_C[redacted]= 0;
            m_b[redacted]= 1;
        }
    }

    pSp[redacted]IP;
    pSp[redacted]rt;
    if [redacted]rt)
    {
        CPl[redacted]IP,
m_C[redacted]t);
        if[redacted]f)
        {
            str[redacted]ker-
>c[redacted]e));
        }
    }
}
```

EXHIBIT C

```
/*****************************************************************************
 *
 * Copyright (c) [redacted] SCEA. All Rights Reserved.
 *
 [redacted]
 *
 *****************************************************************************/

/*****************************************************************************
 *
 *  File Name:      [redacted]
 *
 *  Description:      Implementation of the CAudio class.
 *
 *
 *  Programmers:    Anthony Mai (am) anthony_mai@playstation.sony.com
 *
 *  History:        [redacted]
 *
 *  Notes:          [redacted]
 *
 *****************************************************************************/

#if [redacted]32)
#in[redacted].h>
#in[redacted].h>
#en[redacted]N32)

#in[redacted].h>
#in[redacted].h"
#in[redacted].h"
#in[redacted].h"
#in[redacted].h"
#in[redacted].h"
#in[redacted].h"
#in[redacted].h"

#in[redacted].h"
#in[redacted].h"
#in[redacted].h"


#de[redacted]24

st[redacted] = 0;

vo[redacted]me()
{
    gTi[redacted]();
}
```

```
RT_[redacted]e()
{
    re[redacted]rt);
}

/*******************************************************************************
* Function:          [redacted]
*
* Description:    Convert IP string like "127.0.0.1" to binary value. For example
*                 "127.0.0.1" converts to 0x7f000001.
*
* Returns:        [redacted]
[redacted]
*******************************************************************************/
sta[redacted]IP
(
    RT_[redacted]ng
)
{
    RT_[redacted]=0;
    RT_[redacted]      c;

    if [redacted]g)
    {
        wh[redacted]++)
        {
            if [redacted].')
            {
                ip [redacted]al;
                va[redacted] 0;
            }
            el[redacted]9'))
            {
                va[redacted] [redacted]0');
            }
        }
        ip [redacted]al;
    }

    re[redacted]p;
}


/*******************************************************************************
* Function:          [redacted]
*
* Description:    Convert a binary IP to IP string. For example 0x7f000001 will
*                 be converted to "127.0.0.1".
*
* Returns:        [redacted]
[redacted]
*******************************************************************************/
st[redacted]ng
(
    RT_[redacted]p,
    RT_[redacted]g
)
```

```
{
    RT_[redacted]=0;

    if [redacted]g)
    {
        for [redacted]++)
        {
            i[redacted]0)
            {
                *pIP[redacted].';
            }
            va[redacted]>24;
            if [redacted]00)
            {
                *pIP[redacted]0');
                va[redacted]0;
                go[redacted]t_2;
            }
            e[redacted]= 10)
            {
            d[redacted]2:
                *pI[redacted]0');
                val [redacted]0;
                go[redacted]1;
            }
            e[redacted]
            {
            di[redacted]:
                *pI[redacted]');
            }
            ip [redacted] 8;
        }
        *pI[redacted]00;
    }
}

/*****************************************************************************
* Function:          [redacted]
*
* Description:       [redacted] class constructor
*
* Returns:         [redacted]
*****************************************************************************/
CA[redacted]o() :
    m_[redacted]0),
    m_[redacted]0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted]L),
    m_[redacted]L),
    m_[redacted]L),
    m_[redacted]L),
    m_[redacted]L),
    m_[redacted]L),
    m_[redacted]L),
    m_[redacted]LL),
    m_[redacted]L),
    m_[redacted](0),
    m_[redacted]L),
    m_[redacted]0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted]s(0),
    m_[redacted](0),
    m_[redacted](0),
    m_[redacted](0)
{
    me[redacted]ks));
}


/*****************************************************************************
* Function:        [redacted]
*
* Description:     [redacted] class destructor
*
* Returns:        [redacted]
*****************************************************************************/
[redacted]
{
    Des[redacted]();
    Des[redacted]();
    Cle[redacted]);
}


/*****************************************************************************
* Function:        [redacted]
*
* Description:      Re-start the audio sequence from zero. This must be done when
*                  we first starts to speak. Or starts speaking after a pause.
*
* Returns:        [redacted]
*****************************************************************************/
voi[redacted]ce()
{
    m_n[redacted] 0;
}


/*****************************************************************************
* Function:        [redacted]
*
* Description:      Return a sequence number for audio data. Note it is different
*                  from the sequence number returned by [redacted].
*
* Returns:        [redacted]
```

```
*****************************************************************************/
RT_[redacted]e()
{
    ret[redacted]+;
}


/*****************************************************************************
* Function:       [redacted]
*
* Description:    Initialize the [redacted] object. Initialize the GSM codec and
*                 sample rate converter. Initialize the rt_comm layer. We do
*                 NOT check the validity of parameters passed in.
*
* Returns:        [redacted]
*****************************************************************************/
AUD[redacted]ze
(
    int      [redacted]n,
    int      [redacted]In,
    int      [redacted]n,
    int      [redacted],
    int      [redacted]eOut,
    int      [redacted]ut,
    int      [redacted]t,
    REC[redacted]nc,
    PLA[redacted]unc
)
{
    RT_[redacted]pt;
    RT_[redacted]OK;
    RT_[redacted]6];

    if [redacted]= 0)
    {
        re[redacted]ID;
    }

    Ad[redacted]);

    m_p[redacted]c;
    m_[redacted]nc;

    re[redacted]p();
    if [redacted]et)
    {
        ret[redacted]IC;
    }

    Chn[redacted]UDP;
    Chn[redacted]EN;

    str[redacted]0;
    rt_[redacted]IP);
    m_L[redacted]IP);

    m_L[redacted]rt;
    ret [redacted]te(
        (voi[redacted]n,
        N[redacted],
            n[redacted],
            &C[redacted]
        );
    if [redacted]t)
    {
        re[redacted]C;
    }

    m_p[redacted]l);
    m_p[redacted]r);

    m_p[redacted]M();
    m_p[redacted]M();

    m_p[redacted]is);

    m_S[redacted]ZED;

    ret[redacted]OR;
}

/*******************************************************************************
* Function:        [redacted]
*
* Description:    Clean up before the object is destroyed or re-initialized.
*
* Returns:        [redacted]
*******************************************************************************/
voi[redacted]up()
{
    whi[redacted]0)
    {
        de[redacted]ev);
    }

    if [redacted]n)
    {
        rt_c[redacted]nn);
        m_[redacted]L;
    }

    rt_[redacted]n();

    if [redacted]ce)
    {
        de[redacted]ce;
        m_[redacted]L;
    }

    m_S[redacted] 0;
}
```

```
/*****************************************************************************
 * Function:        [redacted]
 *
 * Description:    Return the local IP. The IP may NOT be the same as what
remote
 *                 machine sees if a network proxy is used. [redacted]
 *
 * Returns:       [redacted].
*****************************************************************************/
uns[redacted]IP()
{
    re[redacted]IP;
}


/*****************************************************************************
 * Function:        [redacted]
 *
 * Description:    Return the local UDP port used.
 *
 * Returns:       [redacted]
*****************************************************************************/
uns[redacted]rt()
{
    re[redacted]rt;
}


/*****************************************************************************
 * Function:        [redacted]
 *
 * Description:    Join a game. The client must know at least one remote client's
 *                 IP and Port to be able to join. That information may be obtained
 *                 by the application from the game server.
 *
 * Returns:       [redacted]
*****************************************************************************/
AUD[redacted]in
(
    uns[redacted]IP,
    uns[redacted]rt
)
{
    AUD[redacted]dr;
    CLI[redacted]in;

    if [redacted]= 0)
    {
        ret[redacted]ID;
    }

    m_p[redacted]n));

    cli[redacted]in);
    cli[redacted]IP;
    cli[redacted]rt;
```

```
        m_[redacted]NED;

        ret[redacted]ge(
            ho[redacted],
            ho[redacted]
            &s[redacted]dr,
            &c[redacted],
            si[redacted]n)
            );
}


voi[redacted]t()
{
    if [redacted] > 0)
    {
        Qui[redacted]el);
    }

    if [redacted]d0)
    {
        m_[redacted]t();
    }

    m_[redacted]NED;
}


/******************************************************************************
* Function:         [redacted]
*
* Description:      Process any incoming data and send out pending outgoing data.
*                 This function must be called by the application periodically
*                 to do the processing.
*
* Returns:        [redacted]
******************************************************************************/
AUD[redacted]e()
{
    // Re[redacted].
    // Proc[redacted]
    // Repe[redacted]
    // forw[redacted]
    AUD[redacted]OR;
    RT_[redacted]P;
    RT_[redacted] 0;
    RT_[redacted]96];
    RT_[redacted] 0;
    CGr[redacted]LL;

    for [redacted]fer
    {
        rec[redacted]rt);

        if [redacted] 0)
        {
            // Er[redacted] it
            ret [redacted]RIC;
            b[redacted];
        }
        if [redacted] 0)
        {
            b[redacted]er.
        }.
        // We [redacted]age
        if ([redacted]ff,
[redacted]nt))
        {
            // Err[redacted] it
            ret [redacted]C;
            b[redacted];
        }
    }

    whi[redacted]OR)
    {
        if [redacted](0))
        {
            ret [redacted]te();
        }

        if [redacted]OR)
        {
            b[redacted];
        }

        if [redacted]el)))
        {
            ret [redacted]e();
        }.

        b[redacted];
    }

    // Pro[redacted]ta
    if [redacted]ULL)
    {
        re[redacted] 0;
        whi[redacted]ff)))
> 0)
        {
            Str[redacted]t);
            if [redacted]f))
            {
                b[redacted];
            }
        }
    }

    // Pro[redacted]a.
    rec[redacted] 0;
    whi[redacted]))) > 0)
    {
        Str[redacted]nt);
    }

    re[redacted]t;
)


/*******************************************************************************
* Function:       [redacted]
*
* Description:     Create a new audio channel. It fails if the channel name has
*                 already been used.
*
* Returns:        [redacted]
*******************************************************************************/
AU[redacted]el
(
    int[redacted]um,
    con[redacted]me
)
{
    CGr[redacted]d0;
    CPl[redacted]er;
    AUD[redacted]dr;
    CRE[redacted]el;

    if [redacted]) ||
        (pC[redacted]L) ||
        (*pC[redacted]0) )
    {
        ret[redacted]ID;
    }

    *pC[redacted] 1;
    whi[redacted]um))
    {
        (*pC[redacted] ++;
    }

    whi[redacted]d0))
    {
        if [redacted]= 0)
        {
            // The [redacted].
            ret[redacted]D;
        }
    }

    if [redacted] 0)
    {
        Qui[redacted]1);
        m_c[redacted] = 0;
    }

    pGr[redacted]m);
    str[redacted]e));
    pGr[redacted]00;
    pTh[redacted]rt);
    pGr[redacted]r);

    cre[redacted]el);
    cre[redacted]l;
    str[redacted]me,
siz[redacted]e));
    cre[redacted]0;

    m_p[redacted]er(
        &se[redacted]r,
        MSG[redacted]EL,
        siz[redacted]el)
        );

    m_c[redacted]el;
    pTh[redacted]el);

    // We [redacted]w.
    re[redacted]ge(
        m_[redacted]P,
        m_[redacted]rt,
        &s[redacted]dr,
        &c[redacted]el,
        si[redacted]el)
        );
}


/*****************************************************************************
* Function:         [redacted]
*
* Description:      Process an incoming package. It could be a control package or
*                   audio data package.
*
* Returns:          [redacted]
*****************************************************************************/
AUD[redacted]ge
(
    RT_[redacted]P,
    RT_[redacted]rt,
    RT_[redacted]er,
    RT_[redacted]ze
)
{
    AUD[redacted]ck;
    CG[redacted]L;
    in[redacted]ex;

    // San[redacted]e.
    if ([redacted]k))
    {
        re[redacted]IC;
    }

    me[redacted]ck));
```

```
if [redacted]OR)
{
    // Beca[redacted]nt.
    th[redacted]mIP;
    th[redacted]ort;
    th[redacted]OR;
    me[redacted]ck));
}

swi[redacted]e)
{
ca[redacted]CM:
ca[redacted]SM:
ca[redacted]10:
    // We i[redacted]s.
    if [redacted]E))))
    {
        ret[redacted]C;
    }
    b[redacted];
de[redacted]:
    if [redacted]k)))
    {
        ret[redacted]IC;
    }
    b[redacted];
}

// Is t[redacted]f?
if [redacted]t))
{
    ret[redacted]OR;
}

pGr[redacted]l);
if [redacted]L)
{
    // This [redacted] to.
    ret[redacted]IC;
}

// Is [redacted]ge?
iIn[redacted]k;
whi[redacted]ck)
{
    if [redacted]ce)
    {
        co[redacted];
    }
    if [redacted]IP)
    {
        co[redacted];
    }
    if [redacted]rt)
    {
            co[redacted];
        }
        if [redacted]e)
        {
            co[redacted];
        }
        if [redacted]el)
        {
            co[redacted];
        }
        if [redacted]th)
        {
            co[redacted];
        }
        // Th[redacted]
        ret[redacted]OR;
    }
    m_p[redacted]ck;
    m_i[redacted]-1);

    if [redacted] &&
        (th[redacted]M) )
    {
        ret[redacted]
[redacted]ze);
    }
    e[redacted]
    {
        ret[redacted]
[redacted]ze);
    }
}


/*******************************************************************************
* Function:         [redacted]
*
* Description:      Send a package to a specific UDP address.
*
* Returns:        [redacted].
*******************************************************************************/
AU[redacted]ge
(
    RT_[redacted]IP,
    RT_[redacted]rt,
    AUD[redacted]dr,
    con[redacted]er,
    RT_[redacted]ze
)
{
    RT_[redacted]ze;
    RT_[redacted]ZE];

    mem[redacted]));
    nSi[redacted]r);

    mem[redacted]ze);
    nSi[redacted]e;

    if [redacted]rt))
    {
        ret[redacted]OR;
    }
    e[redacted]
    {
        ret[redacted]C;
    }
}


/******************************************************************************
* Function:          [redacted]
*
* Description:       Send a chunk of data by UDP to a remote address.
*
* Returns:          [redacted]
******************************************************************************/
in[redacted]To
(
    con[redacted]er,
    RT_[redacted]ze,
    RT_[redacted]IP,
    RT_[redacted]rt
)
{
    RT_[redacted]lt;
    RT_[redacted]0;
    RT_[redacted]6];

    IP[redacted]dr);

      res[redacted]o(
            (v[redacted]n,
            t[redacted],
            (RT[redacted]t,
            pB[redacted]r,
            (RT[redacted]e,
            &[redacted]nt
        );

    if [redacted]OK)
    {
        // Dea[redacted]
        ret[redacted]C;
    }

    ret[redacted]nt;
}


/******************************************************************************
* Function:          [redacted]
*
* Description:       Receive any incoming data on the UDP port.
```

```
*
* Returns:      [redacted]
*******************************************************************************/
int[redacted]om
(
    RT_[redacted]er,
    RT_[redacted]ze,
    RT_[redacted]IP,
    RT_[redacted]rt
)
{
    RT_[redacted]t;
    uns[redacted] [redacted] 0;
    cha[redacted][32];

    *pR[redacted] 0;
    *pR[redacted] 0;

    re[redacted]om(
            (voi[redacted]n,
            fr[redacted]r,
            (RT[redacted]t,
            pB[redacted],
            (RT[redacted]e,
            &r[redacted]t
        );

    if [redacted]OK)
    {
        // Deal [redacted]
        ret[redacted]C;
    }
    el[redacted] 0)
    {
        *pRe[redacted]r);
    }

    re[redacted]t;
}

/*******************************************************************************
* Function:         [redacted]
*
* Description:      Add a new grid which is associate with a specific channel.
*
* Returns:          [redacted]
*******************************************************************************/
[redacted]
(
    int [redacted]l
)
{
    CG[redacted]el);

    if [redacted]0)
    {
        pGr[redacted]0;
        pGr[redacted]ev;
        pGr[redacted]id;
        pGr[redacted]id;
    }
    e[redacted]
    {
        pGr[redacted]id;
        pGr[redacted]id;
        m_p[redacted]d;
    }

    re[redacted];
}

/*******************************************************************************
* Function:        [redacted]
*
* Description:     Find a grid associated with a specific channel number.
*
* Returns:         [redacted]
*******************************************************************************/
[redacted]
(
    int [redacted]l
)
{
    CGr[redacted]0;
    whi[redacted]d)
    {
        if [redacted]el)
        {
            b[redacted];
        }
        e[redacted]
        {
            pGr[redacted]t;
            if [redacted]0)
            {
                pG[redacted]L;
                b[redacted];
            }
        }
    }
    ret[redacted]d;
}

/*******************************************************************************
* Function:        [redacted]
*
* Description:     Remove a grid from the linked list. Caller needs to make sure
*                  it is in the linked list since we don't check. Note we do not
*                  call delete so the caller needs to do the delete.
*
```

```
* Returns:      [redacted]
*******************************************************************************/
[redacted]
(
    CG[redacted]id
)
{
    if [redacted]d)
    {
        if [redacted]d0)
        {
            m_[redacted]LL;
        }
        else
        {
            pGr[redacted]xt;
            pGr[redacted]ev;
        }
    }

    ret[redacted]d;
}

/*******************************************************************************
* Function:       [redacted]
*
* Description:    Cleanup the sample rate converter.
*
* Returns:        [redacted]
*******************************************************************************/
voi[redacted]s()
{
    if [redacted]n)
    {
        del[redacted]In;
        m_p[redacted]LL;
    }
    if [redacted]t)
    {
        del[redacted]ut;
        m_p[redacted]LL;
    }
}

/*******************************************************************************
* Function:       [redacted]
*
* Description:    Clean up the GSM codec.
*
* Returns:        [redacted]
*******************************************************************************/
voi[redacted]s()
{
    if [redacted]n)
    {
        del[redacted]n;
        m_[redacted]L;
    }
    if [redacted]t)
    {
        del[redacted]t;
        m_p[redacted]L;
    }
}


/*****************************************************************************
* Function:         [redacted]
*
* Description:      The audio stream output function. The [redacted] calls this function
*                   when it has received compressed audio data from the network, and
*                   has sequenced the data properly. We do decoding within this
*                   and the decoded data is directly sent to the PLAYFUNC provided
*                   by the application.
*
* Returns:          [redacted]
*****************************************************************************/
int [redacted]put
(
    RT_[redacted]er,
    RT_[redacted]ze
)
{
    RT_[redacted]nk;
    gsm[redacted]0];

    whi[redacted] 0)
    {
        if [redacted] 0)
        {
            one[redacted]ze;
            if [redacted]ze)
            {
                me[redacted]ze);
                m_[redacted]ze;
                b[redacted];
            }
            e[redacted]e
            {
                me[redacted]nk);
                pB[redacted]nk;
                c[redacted]nk;
                St[redacted]);
                m_[redacted]ta);
                m_[redacted]e();
                m_[redacted]ZE;
                if [redacted]y)
                {
                    m_[redacted]ZE);
                }
                nD[redacted]ZE;
                m_[redacted]E;
            }
        }
        el[redacted]ZE)
        {
            me[redacted]ze);
            m_[redacted]e;
            b[redacted];
        }
        e[redacted]
        {
            St[redacted]();
            m_[redacted]ta);
            m_[redacted]e();
            m_[redacted]IZE;
        .   if [redacted]y)
            {
                m_[redacted]ZE);
            }
            pB[redacted]ZE;
            cb[redacted]ZE;
            nD[redacted]ZE;
        }
    }

    re[redacted]ed;
}


/*****************************************************************************
* Function:          [redacted]
*
* Description:     Audio stream input function. Application calls this function
*                  periodically when it has audio data available (from Microphone).
*                  Flow control (make sure we stream the right amount of data in
*                  the right amount of time) is done by the caller. Input size 0
*                  has special meanings: It signify the end of streaming input.
*
* Returns:         [redacted]
*****************************************************************************/
int [redacted]ut
(
    RT_[redacted]er,
    RT_[redacted]ze
)
{
    RT_[redacted]nk;
    RT_[redacted]2];
    CGr[redacted]d();
    AU[redacted]dr;

    if [redacted]LL)
    {
        ret[redacted]ed;
    }

    aud[redacted]SM;
    aud[redacted]IP;
```

```
        aud[redacted]rt;
        aud[redacted]el;

        if [redacted] 0)
        {
            au[redacted]e();
            au[redacted] = 0;
            pS[redacted]ge(
                m_[redacted]P,
                m_[redacted]t,
                &a[redacted]r,
                gs[redacted]a,
                au[redacted]th
                );

            Res[redacted]e();
        }
        el[redacted] > 0)
        {
            if [redacted] > 0)
            {
                on[redacted]ize;
                if [redacted]ze)
                {
                    mem[redacted]ze);
                    m_I[redacted]e;
                    b[redacted];
                }
                e[redacted]
                {
                    mem[redacted]nk);
                    pBu[redacted]nk;
                    cbS[redacted]nk;
                    m_I[redacted]nk;

                    St[redacted]e();
                    m_[redacted]ta);
                    m_p[redacted]ZE]),
&([redacted]ZE]));
                    m_n[redacted]me();
                    m_n[redacted]ZE*2;

                    // Sen[redacted]id
                    au[redacted]ce();
                    au[redacted]*2;
                    pSp[redacted]e(
                        m_[redacted]P,
                        m_[redacted]rt,
                        &[redacted]r,
                        gs[redacted]a,
                        au[redacted]th
                        );

                    nEn[redacted]*2;
                    m_I[redacted]*2;
                }
            }
        el[redacted]*2))
        {
            mem[redacted]ze);
            m_I[redacted]ze;
            b[redacted]
        }
        e[redacted]
        {
            St[redacted]();
            m_p[redacted]a);
            m_p[redacted]
[redacted]IZE]));
            m_n[redacted]me();
            m_n[redacted]*2;

            // Send[redacted]id
            au[redacted]ce();
            au[redacted]*2;
            pSp[redacted]ge(
                m_[redacted]P,
                m_[redacted]rt,
                &[redacted]r,
                gs[redacted]a,
                au[redacted]th
                );

            pBu[redacted]*2;
            cbS[redacted]*2;
            nEn[redacted]*2;
        }
    }

    ret[redacted]d;
}


/*********************************************************************************
* Function:         [redacted]
*
* Description:      Return pointer to the audio grid we are currently allowed to
*                 speak.
* Returns:        [redacted]
*********************************************************************************/
[redacted]
{
    CGr[redacted]L;
    if [redacted] 0)
    {
        if [redacted]L)
        {
            if ([redacted]d())
            {
                pGr[redacted]L;
            }
            e[redacted]
            {
                //OK,[redacted].
            }
        }
    }

    ret[redacted];
}


/*******************************************************************************
* Function:         [redacted]
*
* Description:     Join a channel by the channel number. The channel number must
*                  be none-zero.
*
* Returns:         [redacted]
*******************************************************************************/
[redacted]
(
    int [redacted]l
)
{
    AUD[redacted]OR;
    AUD[redacted]dr;
    CLI[redacted]el;

    if [redacted] 0)
    {
        Qu[redacted]el);
    }
    if ([redacted] 0)
    {
        CGr[redacted]l);

        joi[redacted]el;
        joi[redacted]IP;
        joi[redacted]rt;

        m_[redacted]r(
            &s[redacted]r,
            MS[redacted]NEL,
            si[redacted]el)
            );

        ret [redacted]e(
            m_[redacted]P,
            m_[redacted]rt,
            &s[redacted]r,
            &jo[redacted]el,
            si[redacted]el)
            );

        if [redacted]OR)
        {
            CP[redacted]
[redacted]ort);
```

```
            m_c[redacted]nel;

            pTh[redacted]el);

            if ([redacted]LL)
            {
                pG[redacted]er);
            }

            m_S[redacted]NEL;
        }
    }

    re[redacted]t;
}


/******************************************************************************
 * Function:       [redacted]
 *
 * Description:    Quit a specific channel by channel number. We check to be sure
 *                 we are currently in the channel before sending the quit message
 *                 out. The channel number must be none-zero.
 *
 * Returns: [redacted]
 ******************************************************************************/
AUD[redacted]el
(
    in[redacted]el
)
{
    AUD[redacted]OR;
    AUD[redacted]dr;
    CLI[redacted]el;
    CGr[redacted]id;

    if [redacted] 0) ||
        (nC[redacted]el) )
    {
        re[redacted]IC;
    }

    qui[redacted]el;
    qui[redacted]IP;
    qui[redacted]rt;

    m_p[redacted]er(
        &se[redacted]r,
        MS[redacted]EL,
        si[redacted]l)
        );

    re[redacted]ge(
        m_[redacted]P,
        m_[redacted]ort,
        &s[redacted]r,
            &q[redacted]el,
            si[redacted]nel)
            );

        if [redacted]OR)
        {
            pGr[redacted]el);
            if [redacted]LL)
            {
                pGr[redacted]t);
                pGr[redacted]on();
            }
            m_c[redacted] = 0;
            m_p[redacted]el);

            m_S[redacted]EL;
        }

        re[redacted]t;
}

/*******************************************************************************
* Function:          [redacted]
*
* Description:    Enumerate all existing channels, not just those we are in. The
*                 enumeration continues until all channels have run through, or
*                 the enumeration function returns zero.
*
* Returns:        [redacted]
*******************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
)
{
    CGr[redacted]d0;

    if [redacted]L)
    {
        ret[redacted]IC;
    }

    whi[redacted]LL)
    {
        CH[redacted]fo;

        cha[redacted]el;
        str[redacted]e,
si[redacted]me));
        cha[redacted]d-
>m[redacted]nt();

        if [redacted]fo))
        {
            b[redacted]
        }

        pGr[redacted]xt;
        if [redacted]d0)
        {
            pGr[redacted]LL;
        }
    }

    ret[redacted]OR;
}


/*************************************************************************************
* Function:          [redacted]
*
* Description:         Enumerate all existing players (in channel 0). The enumeration
*                    continues until all players have run through, or the enumeration
*                    function returns zero.
*          .
* Returns:           [redacted]
*************************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
)
{
    CPl[redacted]LL;
    CPl[redacted]LL;

    if [redacted]LL))
    {
        ret[redacted]C;
    }

    pPl[redacted]er();

    whi[redacted]LL)
    {
        CLI[redacted]fo;

        cli[redacted]_IP;
        cli[redacted]rt;
        str[redacted]e(),
si[redacted]me));
        cli[redacted]el;

        if [redacted]fo))
        {
            b[redacted];
        }

        pPl[redacted]t;
        if [redacted]r0)
        {
            pP[redacted];
        }
    }

    re[redacted]OR;
}


/*******************************************************************************
 * Function:       [redacted]
 *
 * Description:     Enumerate all existing players in a specific channel. Enumeration
 *                 continues until all players have run through, or the enumeration
 *                 function returns zero.
 *
 * Returns:        [redacted]
 *******************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
    int[redacted]ted.
)
{
    CGr[redacted]el);
    CPl[redacted]LL;
    CPl[redacted]LL;
    CPl[redacted]LL;

    if [redacted]LL))
    {
        ret[redacted]IC;
    }

    pRe[redacted]ef();

    whi[redacted]LL)
    {
        CLI[redacted]fo;

        pPl[redacted]er;
        cli[redacted]_IP;
        cli[redacted]rt;
        strncpy(clientInfo.clientName, pPlayer->GetName(),
siz[redacted]me));
        cli[redacted]el;

        if [redacted]fo))
        {
            b[redacted];
        }

        pRe[redacted]t;
        if [redacted]f0)
        {
            pR[redacted]L;
        }
    }
```

```
        ret[redacted]OR;
    }


/*******************************************************************************
 * Function:        [redacted]
 *
 * Description:     Enumerate all players in a specific channel that we directly
 *                  connect with. Enumeration continues until all players have run
 *                  through, or the enumeration function returns 0.
 *
 * Returns:         [redacted]
 ******************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
    int[redacted]ated.
)
{
    CGr[redacted]el);

    if [redacted]LL))
    {
        ret[redacted]IC;
    }
    e[redacted]
    {
        ret[redacted]ta);
    }
}


/*******************************************************************************
 * Function:        [redacted]
 *
 * Description:     Create and add a new CPlayer object.
 *
 * Returns:         [redacted]
 ******************************************************************************/
[redacted]
(
    RT_[redacted]IP,
    RT_[redacted]rt,
    RT_[redacted]el
)
{
    ret[redacted]el);
}


/*******************************************************************************
 * Function:        [redacted]
 *
 * Description:     Remove and delete an existing [redacted] object associated with a
 *                  specific player.
```

```
*
* Returns:        [redacted]
*********************************************************************************/
vo[redacted]er
(
    RT_U[redacted]rt
)
{
    CPl[redacted]rt);

    if [redacted]r)
    {
        if [redacted] > 0)
        {
            CGr[redacted]el);
            if [redacted]d)
            {
                pGr[redacted]t);
            }
        }
        if [redacted]d0)
        {
            m_[redacted]rt);
        }

        m_P[redacted]rt);
    }
}

/*********************************************************************************
* Function:        [redacted]
*
* Description:     Update and returns the status flag. If a remote client is
*                  speaking, the remote client info is returned using [redacted].
*
* Returns:         [redacted]
*********************************************************************************/
[redacted]
(
    CLI[redacted]er
)
{
    CGr[redacted] 0;

    if [redacted] > 0)
    {
        pGr[redacted]el);
    }

    if [redacted]er)
    {
        pGr[redacted]er);
    }
    e[redacted]
    {
        pSp[redacted]00;
        pSp[redacted] 0;
        pSp[redacted] 0;
        pSp[redacted] 0;
    }

    if [redacted]id())
    {
        m_S[redacted]LED;
    }
    e[redacted]
    {
        m_S[redacted]ED;
    }

    re[redacted]us;
}


/*****************************************************************************
* Function:         [redacted]
*
* Description:     Re-initialize the [redacted] decoder when the source of streaming
*        .         audio changes.
*
* Returns:         [redacted]
*****************************************************************************/
voi[redacted]
{
    if [redacted]ut)
    {
        m_[redacted]it();
    }
}


/*****************************************************************************
* Function:         [redacted]
*
* Description:     Return accumulative average CPU load % for encoding & decoding
*
* Returns:         [redacted]
*****************************************************************************/
vo[redacted]oad
(
    dou[redacted]de,
    dou[redacted]de
)
{
    if [redacted]es)
    {
        *pE[redacted]tes;
    }
    e[redacted]
    {
        *pEn[redacted] = 0;
    }
    if [redacted]es)
```

```
        {
            *pDe[redacted]tes;
        }
        e[redacted]
        {
            *pD[redacted] 0;
        }
}
```

EXHIBIT C-1

Exhibit C1

```
/*************************************************************************
**
* Function:         [redacted]
*
* Description:     Create a new audio channel. [redacted]
*
*               [redacted]
*
* Returns:        [redacted]
*************************************************************************
*/
AU[redacted]el
(
    int[redacted]um,
    con[redacted]me
)
{
    CGr[redacted]d0;
    CPl[redacted]er;
    AUD[redacted]dr;
    CRE[redacted]el;

    if [redacted]) ||
        (pC[redacted]L) ||
        (*pC[redacted]0) )
    {
        ret[redacted]ID;
    }

    *pC[redacted] 1;
    whi[redacted]um))
    {
        (*pC[redacted] ++;
    }

    whi[redacted]d0))
    {
        if [redacted]= 0)
        {
            // The [redacted].
            ret[redacted]D;
        }
    }

    if [redacted] 0)
    {
        Qui[redacted]l);
        m_c[redacted] = 0;
    }

    pGr[redacted]m);
    str[redacted]e));
    pGr[redacted]00;

    pTh[redacted]rt);
    pGr[redacted]r);

    cre[redacted]el);
```

Exhibit C1

```
        cr[redacted]l;
        st[redacted]me,
    siz[redacted]e));
        cre[redacted]0;

        m_p[redacted]er(
            &se[redacted]r,
            MSG[redacted]EL,
            siz[redacted]el)
            );

        m_c[redacted]el;
        pTh[redacted]el);

        // We [redacted]w.
        re[redacted]ge(
            m_[redacted]P,
            m_[redacted]rt,
            &s[redacted]dr,
            &c[redacted]el,
            si[redacted]el)
            );
    }
```

Exhibit C2

```
/*****************************************************************************
**
* Function:         [redacted]
*
* Description:    Process an incoming package. [redacted]
*              [redacted]
*
* Returns:        [redacted]
*****************************************************************************
*/
AUD[redacted]ge
(
    RT_[redacted]P,
    RT_[redacted]rt,
    RT_[redacted]er,
    RT_[redacted]ze
)
{
    AUD[redacted]ck;
    CG[redacted]L;
    in[redacted]ex;

    // San[redacted]e.
    if ([redacted]k))
    {
        re[redacted]IC;
    }

    me[redacted]ck));

    if [redacted]OR)
    {
        // Bec[redacted]nt.
        th[redacted]mIP;
        th[redacted]ort;
        th[redacted]OR;
        me[redacted]ck));
    }

    swi[redacted]e)
    {
    ca[redacted]CM:
    ca[redacted]SM:
    ca[redacted]10:
        // We i[redacted]s.
        if [redacted]
[redacted]E))))
        {
            ret[redacted]C;
        }
        b[redacted];
    de[redacted]:
        if [redacted]k)))
        {
            ret[redacted]IC;
        }
        b[redacted];
```

<u>Exhibit C2</u>

```
    }

    // Is t[redacted]f?
    if [redacted]
[redacted]t))
    {
        ret[redacted]OR;
    }

    pGr[redacted]l);
    if [redacted]L)
    {
        // This [redacted] to.
        ret[redacted]IC;
    }

    // Is [redacted]ge?
    iIn[redacted]k;
    whi[redacted]ck)
    {
        if [redacted]ce)
        {
            co[redacted];
        }
        if [redacted]IP)
        {
            co[redacted];
        }
        if [redacted]rt)
        {
            co[redacted];
        }
        if [redacted]e)
        {
            co[redacted];
        }
        if [redacted]el)
        {
            co[redacted];
        }
        if [redacted]th)
        {
            co[redacted];
        }
        // Th[redacted]
        ret[redacted]OR;
    }
    m_p[redacted]ck;
    m_i[redacted]-1);

    if [redacted] &&
        (th[redacted]M) )
    {
        ret[redacted]
[redacted]ze);
    }
    e[redacted]
    {
        ret[redacted]
[redacted]ze);
    }
}
```

<u>Exhibit C3</u>

```
/*********************************************************************************
* Function:       [redacted]
*
* Description:    Enumerate all existing channels, not just those we are in. The
*                 enumeration continues until all channels have run through, or
*                 the enumeration function returns zero.
*
* Returns:        [redacted]
********************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
)
{
    CGr[redacted]d0;

    if [redacted]L)
    {
        ret[redacted]IC;
    }

    whi[redacted]LL)
    {
        CH[redacted]fo;

        ch[redacted]el;
        str[redacted]e,
si[redacted]me));
        ch[redacted]d-
>m[redacted]nt();

        if [redacted]fo))
        {
            b[redacted]
        }

        pGr[redacted]xt;
        if [redacted]d0)
        {
            pGr[redacted]LL;
        }
    }

    ret[redacted]OR;
}
```

Exhibit C4

```
/*****************************************************************************
* Function:         [redacted]
*
* Description:    Enumerate all existing players (in channel 0). The enumeration
*                 continues until all players have run through, or the enumeration
*                 function returns zero.
*
* Returns:        [redacted]
*****************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
)
{
    CPl[redacted]LL;
    CPl[redacted]LL;

    if [redacted]LL))
    {
        ret[redacted]C;
    }

    pPl[redacted]er();

    whi[redacted]LL)
    {
        CLI[redacted]fo;

        cli[redacted]_IP;
        cli[redacted]rt;
        str[redacted]e(),
si[redacted]me));
        cli[redacted]el;

        if [redacted]fo))
        {
            b[redacted];
        }

        pPl[redacted]t;
        if [redacted]r0)
        {
            pP[redacted];
        }
    }

    re[redacted]OR;
```

Exhibit C5

```
/*****************************************************************************
 * Function:         [redacted]
 *
 * Description:   Enumerate all existing players in a specific channel. Enumeration
 *                continues until all players have run through, or the enumeration
 *                function returns zero.
 *
 * Returns:       [redacted]
 *****************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
    int [redacted]ted.
)
{
    CGr[redacted]el);
    CPl[redacted]LL;
    CPl[redacted]LL;
    CPl[redacted]LL;

    if [redacted]LL))
    {
        ret[redacted]IC;
    }

    pRe[redacted]ef();

    whi[redacted]LL)
    {
        CLI[redacted]fo;

        pPl[redacted]er;
        cli[redacted]_IP;
        cli[redacted]rt;
        strncpy(clientInfo.clientName, pPlayer->GetName(),
siz[redacted]me));
        cli[redacted]el;

        if [redacted]fo))
        {
            b[redacted];
        }

        pRe[redacted]t;
        if [redacted]f0)
        {
            pR[redacted]L;
        }
    }

    ret[redacted]OR;
}
```

Exhibit C6

```
/*****************************************************************************
* Function:        [redacted]
*
* Description:       Enumerate all players in a specific channel that we directly
*                  connect with. Enumeration continues until all players have run
*                  through, or the enumeration function returns 0.
*
* Returns:         [redacted]
*****************************************************************************/
[redacted]
(
    ENU[redacted]ion
    uns[redacted]ion.
    int[redacted]ated.
)
{
    CGr[redacted]el);

    if [redacted]LL))
    {
        ret[redacted]IC;
    }
    e[redacted]
    {
        ret[redacted]ta);
    }
}
```

Exhibit C7

```
/*****************************************************************************
* Function:        [redacted]
*
* Description:     Join a channel by the channel number. The channel number must
*                  be none-zero.
*
* Returns: [redacted]
******************************************************************************/
[redacted]
(
    int [redacted]l
)
{
    AUD[redacted]OR;
    AUD[redacted]dr;
    CLI[redacted]el;

    if [redacted] 0)
    {
        Qu[redacted]el);
    }
    if ([redacted] 0)
    {
        CGr[redacted]l);

        joi[redacted]el;
        joi[redacted]IP;
        joi[redacted]rt;

        m_[redacted]r(
            &s[redacted]r,
            MS[redacted]NEL,
            si[redacted]el)
            );

        ret [redacted]e(
            m_[redacted]P,
            m_[redacted]rt,
            &s[redacted]r,
            &jo[redacted]el,
            si[redacted]el)
            );

        if [redacted]OR)
        {
            CP[redacted]
[redacted]ort);

            m_c[redacted]nel;

            pTh[redacted]el);

            if ([redacted]LL)
            {
                pG[redacted]er);
            }
```

```
                m_S[redacted]NEL;
            }
        }

        re[redacted]t;
}
```

Exhibit C8

```
/******************************************************************************
* Function:       [redacted]
*
* Description:    Create and add a new CPlayer object.
*
* Returns:      [redacted]
*****************************************************************************/
[redacted]
(
    RT_[redacted]IP,
    RT_[redacted]rt,
    RT_[redacted]el
)
{
    ret[redacted]el);
}
```